Hadoop: How to plan a cluster?

Ok, you have decided to setup a Hadoop cluster for your business.

Next step now, planning the cluster… But Hadoop is a complex stack and you might have many questions:

- HDFS deals with replication and Map Reduce create files… How can I plan my storage needs?
- How to plan my CPU needs?
- How to plan my memory needs? Should I consider different needs on some nodes of the cluster?
- I heard that Map Reduce moves its job code where the data to process is located… What does it involve in terms of network bandwidth?
- At which point and how far should I consider what the final users will actually process on the cluster during my planning?

That is what we are trying to make clearer in this article by providing explanations and formulas in order to help you to best estimate your needs.

Some important aspects of the Hadoop architecture

Planning a Hadoop cluster requires a minimum knowledge the Hadoop architecture.

The distributed computation

At his heart, Hadoop is a distributed computation platform. This platform's programming model is Map Reduce.

In order to be efficient, Map Reduce has two prerequisites:

- Datasets must be splitable in smaller and independant blocks
- Data locality: means that the code must be moved where the data lies, not the opposite.

The first prerequisite depends on both the type of input data which feeds the cluster and what we want to do with it.

The second prerequisite involves having a distributed storage system which exposes where exactly data is stored and allows the execution of code on any storage node.

This is where HDFS is useful.

Hadoop is a Master / Slave architecture:

- The *JobTracker* (*ResourceManager*in Hadoop 2)
  - Monitor jobs that are running on the cluster
  - Needs a lot of memory and CPU (*memory bound* and *cpu bound*)
- The *TaskTracker* (*NodeManager* + *ApplicationMaster*in Hadoop 2)
  - Runs tasks of a jobs on each node of the cluster. Which means *Maps* and *Reduces*
  - Its jobs need a lot of memory and CPU (*memory bound* and *cpu bound*)

The critical component in this architecture is the *JobTracker/ResourceManager*.

The distributed storage

HDFS is a distributed storage filesystem. It runs on top of another filesystem like ext3 or ext4.

In order to be efficient, HDFS must satisfy the following prerequisites:

- Hard drives with a high throughput
- An underlying filesystem which supports the HDFS read and write pattern: one big read or write at a time (64MB, 128MB or 256MB)
- Network fast enough to cope with intermediate data transfer and block replication

HDFS is a Master / Slave architecture:

- The *NameNode* and the *Secondary NameNode*
  - Stores the filesystem meta informations (directory structure, names, attributes and file localization) and ensures that blocks are properly replicated in the cluster
  - It needs a lot of memory (*memory bound*)
- The *DataNode*
  - Manages the state of an HDFS node and interacts with its blocks
  - Needs a lot of I/O for processing and data transfer (*I/O bound*)

The critical components in this architecture are the *NameNode* and the *Secondary NameNode*.

These are two distinct but complementary architectures.

It is possible to not use HDFS with Hadoop. *Amazon* with their Elastic MapReduce for example rely on their own storage offer, *S3* and a desktop tool like *KarmaSphere Analyst* embeds Hadoop with a local directory instead of HDFS.

Some important technicals facts to plan a cluster

No need to be an Hadoop expert but the following few facts are good to know when it comes to cluster planning.

How HDFS manages its files

HDFS is optimized for the storage of large files. You write the file once and access it many times.

In HDFS, a file is split into several blocks. Each block is asynchronously replicated in the cluster. Therefore, the client sends its files once and the cluster takes care of replicating its blocks in the background.

A block is a contigous area, a blob of data on the underlying filesystem, Its default size is 64MB but it can be extended to 128MB or even 256MB, depending on your needs.

The block replication, which has a default factor of 3, is useful for two reasons:

- Ensure data recovery after the failure of a node. Hard drives used for HDFS must be configured in JBOD, not RAID
- Increase the number of maps that can work on a bloc during a MapReduce job and therefore speedup processing

From a network standpoint, the bandwith is used at two moments:

- During the replication following a file write
- During the balancing of the replication factor when a node fails

How the NameNode manages the HDFS cluster

The NameNode manages the meta informations of the HDFS cluster.

This includes meta informations (filenames, directories, …) and the location of the blocks of a file.

The filesystem structure is entirely mapped into memory.

In order to have persistence over restarts, two files are also used:

- a *fsimage* file which contains the filesystem metadata
- the *edits* file which contains a list of modifications performed on the content of *fsimage*.

The in memory image is the merge of those two files.
When the NameNode starts, it first loads *fsimage* and then applies the content of *edits* on it to recover the latest state of the filesystem.
An issue would be that over time, the *edits* file keeps growing undefinitely and ends up by:

- consuming all disk space
- slowdown restarts

The *Secondary NameNode* role is to avoir this issue by regularly merging *edits* with *fsimage*, thus pushing a new*fsimage* and resetting the content of *edits.* The trigger for this compaction process is configurable. It can be:

- The number of transactions performed on the cluster
- The size of the *edits* file
- The elapsed time since the last compaction

The following formula can be applied to know how much memory a *NameNode* needs:
<needed memory> = <total storage size in the cluster in MB> / <Size of a block in MB> / 1000000
In other words, a rule of thumb is to consider that a *NameNode* needs about 1GB / 1 million blocks.

How to determine my sizing needs?

Determine storage needs

Storage needs are split into three parts:

- Shared needs
- *NameNode* and *Secondary NameNode* specific needs
- *DataNode* specific needs

Shared needs

Shared needs are already known since it covers:

- The OS partition
- The OS logs partition

Those two partitions can be setup as usual.

NameNode and Secondary NameNode specific needs

The *Secondary NameNode* must be identical to the *NameNode*. Same hardware, same configuration.
A 1TB partition should be dedicated to files written by both the *NameNode* and the *Secondary NameNode.* This is large enough so you won't have to worry about disk space as the cluster grows.
If you want to be closer to the actual occupied size, you need to take into account the parameters of the*NameNode* we explained above (a combination of the trigger for the compaction, the maximum fsimage size and the *edits* size*)* and to multiply this result by the number of checkpoints you want to be retained.

In any case, the *NameNode* must have an NFS mount point to a secured storage among its *fsimage* and *edits*directories.
This mount point has the same size than the local partition
for *fsimage* and *edits* mentionned above.
The storage of the *NameNode* and the *Secondary NameNode* is typically performed on RAID configuration.

DataNode specific needs

Hardware requirements for *DataNode*s storage is:

- *SAS 6Gb/s controller configured in JBOD (Just a Bunch of Disk)*
- *SATA II 7200 rpm hard drives between 1Tb and 3Tb*

Do not use RAID on a DataNode. HDFS provides its own replication mecanism.

The number of hard drive can vary depending on the total desired storage capacity.

A good way to determine the latter is to start from the planned data input of the cluster.

It is also important to note that for every disk, 30% of its capacity is reserved to non HDFS use.

Let's consider the following hypothesis:

- Daily data input: 100Gb
- HDFS replication factor: 3
- Non HDFS reserved space per disk: 30%
- size of a disk: 3Tb

With these hypothesis, we are able to determine the storage needed and the number of *DataNodes.*
Therefore we have:

- Storage space used by daily data input : <daily data input> * <replication factor> = 300GB
- Size of a hard drive dedicated to HDFS : <Size of the hard drive > * (1 – <Non HDFS reserved space per disk>) = 2.1TB
- Number of *DataNodes* after 1 year (no monthly growth) : <storage space used by daily data input> * 365 / <HDFS reserved space in a disk> = 100TB / 2.1TB = 48 *DataNodes*

Two important elements are not included here:

- The monthly growth of the data input
- The ratio of data generated by jobs processing a data input

These informations depend on the needs of your business units and it must be taken into account in order to determine storage needs.

Determine your CPU needs

On both *NameNode* and *Secondary NameNode*, 4 physical cores running at 2Ghz will be enough.
For *DataNodes*, two elements help you to determine your CPU needs:

- The profile of the jobs that are going to run
- The number of jobs you want to run on each *DataNode*

Job profile

Roughly, we consider that a *DataNode* can perform two kind of jobs: *I/O intensive* and *CPU intensive*.

I/O intensive jobs

These jobs are *I/O bound*.
For example:

- indexing
- search
- clustering
- decompression
- data import/export

Here, a CPU running between 2Ghz and 2.4Ghz is enough.

CPU intensive jobs

These jobs are *CPU bound*.
For example:

- machine learning
- statistics
- semantic analysis
- language analysis

Here, a CPU running between 2.6Ghz and 3Ghz is enough.

The number of jobs

The number of physical cores determine the maximum number of jobs that can run in parallel on a *DataNode*. It is also important to keep in mind that there is a distribution between Map and Reduce tasks on *DataNodes* (typically 2/3 Maps and 1/3 Reduces).
To determine you needs, you can use the following formula:

*(<number of physical cores> – 1) * 1.5 = <maximum number of tasks>*
or, if you prefer to start from the number of tasks and adjust the number of cores according to it:

(<maximum number of tasks> / 1.5) + 1 = <number of physical cores>
The number 2 keeps 2 cores away for both the *TaskTracker* (MapReduce) and *DataNode* (HDFS) processes.
the number 1.5 indicates that a physical core, due to hyperthreading, might process more than one job at the same time.

Determine your memory needs

This is a two step process:

- Determine the memory of both *NameNode* and *Secondary NameNode*
- Determine the memory of *DataNodes*

In both cases, you should use DDR3 ECC memory.

Determine the memory of both *NameNode* and *Secondary NameNode*

As explained above the *NameNode* manages the HDFS cluster metadata in memory. The memory needed for the*NameNode* process and the memory needed for the OS must be added to it.
The *Secondary NameNode* must be identical to the *NameNode*.
Given these informations we have the following formula:

<Secondary NameNode memory> = <NameNode memory> = <HDFS cluster management memory> + <2GB for the NameNode process> + <4GB for the OS>

Determine the memory of *DataNodes*

The memory needed for a *DataNode* is determined depending on the profile of jobs which will run on it.
For *I/O bound* jobs, between 2GB and 4GB per physical core.
For *CPU bound* jobs, between 6GB and 8GB per physical core.
In both cases, the following must be added:

- 2GB for the *DataNode* process which is in charge of managing HDFS blocks
- 2GB for the *TaskTracker* process which is in charge of managing running tasks on the node
- 4GB for the OS

Which leads to the following formulas:

*<DataNode memory for I/O bound profile> = 4GB * <number of physical cores> + <2GB for the DataNode process> + <2GB for the TaskTracker process> + <4GB for the OS>*
*<DataNode memory for CPU bound profile> = 8GB * <number of physical cores> + <2GB for the DataNode process> + <2GB for the TaskTracker process> + <4GB for the OS>*

Determine your network needs

The two reasons for which Hadoop generates the most network traffic are:

- The shuffle phase during which Map tasks outputs are sent to the Reducer tasks
- Maintaining the replication factor (when a file is added to the cluster or when a *DataNode* is lost)

In spite of it, network transfers in Hadoop follow an East/West pattern which means that even though orders come from the *NameNode*, most of the transfers are performed directly between *DataNodes*.
As long as these transfers do not cross the rack boundary, it is not a big issue and Hadoop does its best to perform only such transfers.

However, inter-rack transfers are sometimes needed, for example for the second replica of an HDFS block.

This is complex subject but as a rule of thumb, you should:

- Use a Spine Fabric network topology
    - Better throughput
    - Better resiliency to failures
- Avoid oversubscribtion of switches
    - A 1Gb 48 ports top of rack switch must have 5 ports at 10Gb on distribution switches
    - Avoids network slowdown for a cluster under heavy load (jobs + data input)
- If the cluster is *I/O bound*or if you plan to perform data input on recurrent data input which sature the 1GB and which cannot be performed outside of office hours, you should use:
    - 10Gb Ethernet intra rack
    - N x 10Gb Ethernet inter rack
- If the cluster is *CPU bound*, you should use:
    - 2 x 1Gb Ethernet intra rack
    - 10Gb Ethernet inter rack